Patent

UNITED STATES UTILITY PATENT APPLICATION

FOR

A METHOD AND APPARATUS FOR MANAGING A NETWORK

INVENTORS:

MARK E. EPSTEIN
ROLAND C. DOWDESWELL
MICHAEL F. CUDDY
C. ERIK BERLS

PREPARED BY:

JUDITH A. SZEPESI
BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1026

(408) 720-8300

ATTORNEY'S DOCKET NO. 005389.P001

"Express Mail" mailing label number: EL371006680US
Date of Deposit: December 4, 2001
I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service on the date indicated above and that this paper or fee has been addressed to:
US Patent and Trademark Office, P.O. Box 2327, Arlington VA 22202, Box Patent Application.

Pamala Stephenson
(Typed or printed name of person mailing paper or fee)

Pamala Stephenson Dec 4, 2001
(Signature of person mailing paper or fee) Date Signed

A METHOD AND APPARATUS FOR MANAGING A NETWORK

FIELD OF THE INVENTION

[001] The present invention relates to network control, and more specifically, to controlling a network through distributed control points.

BACKGROUND

[002] There are many ways of controlling a network. However, most of these ways do not cope gracefully with an expanding network, do not successfully distribute work, and/or do not maintain security during the control process.

[003] For example, one prior art method of controlling a network is to physically install new systems using a trusted installer. The installer would have a key or other security mechanism that would be added to the new system on the network. The network then would accept the new system, and would treat it as a trusted connection. If the system on the network needed to be upgraded, a human would have to go out to the system, and upgrade it manually. While this system functions, it becomes difficult if not impossible to maintain when the number of systems coupled the network grows. Furthermore, if the network is a distributed system -- with computers and other devices at locations all over the world -- each location would have to have an actual human able to upgrade the local system. Additionally, it becomes almost impossible to upgrade all systems at the same time, without taking down the network. Furthermore, security may be compromised at any of the multiple locations, and this would not be known until a human verified the security of the system. Therefore, an automated network maintenance and control system would be advantageous.

SUMMARY OF THE INVENTION

[004] A method and apparatus to manage a plurality of devices is described. The apparatus comprises a control server to generate a job to update a device, and a control point to establish a secure communication with the control server, and to receive the job. The control point further to establish communication with the device in accordance with a maintenance schedule, and to update the device using the JobScript found within the job.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

Figure 1 is a block diagram of one embodiment of a network with control servers, control points, and devices.

Figure 2 is a block diagram of one embodiment of a computer system that may be used with the present invention.

Figure 3 is a block diagram of one embodiment of a control server.

Figure 4 is a block diagram of one embodiment of a control point.

Figure 5 illustrates one embodiment of the hierarchical structure of the data store.

Figure 6A is a flowchart of one embodiment of installing a control point, from the control server's perspective.

Figure 6B is a flowchart of one embodiment of installing a control point, from the control point's perspective.

Figure 7A is a flowchart of one embodiment communication between the control server and the control point, from the control server's perspective.

Figure 7B is a flowchart of one embodiment of communication between the control server and the control point, from the control point's perspective.

Figure 7C illustrates one embodiment of establishing connection to the control server, including control server failover, from the perspective of the control point.

Figure 8 is a flowchart of one embodiment of updating a device from a control point.

Figure 9 illustrates one embodiment of command flow-through from the data store to a device.

Figure 10 illustrates one embodiment of generating the data sent by the control server to the control point.

Figure 11 illustrates one embodiment of a device module.

Figure 12 illustrates one embodiment of the history/versioning process.

DETAILED DESCRIPTION

[005] A method and apparatus for network control is described. The system deploys remote control points, which control the various devices that are part of the network. This system may be used, for example, to rapidly provision network elements, audit configurations of network elements, provide fine-grained security access control across the network fabric, and rapidly configure and maintain Virtual Private Network, firewall, and router configurations on the network. The system may also be used within a corporate environment to control all of the devices that are on the network.

[006] The system permits access to the data store using a Command Line Interface (CLI) and a graphical user interface (GUI). The underlying database, for one embodiment, is a SQL database. For one embodiment, the data store includes a schema translator that presents the underlying SQL database as a hierarchical data store.

[007] Using the CLI, GUI, or service modules, the user may make changes in the data store. Service modules permit functionality to be defined and "plugged in" into the control server. Service modules may be designed by users, and may interface with the control server to provide additional usability or alternative interface mechanisms. The user may also create action scripts which use the CLI. Action scripts may have additional linked scripts or commands, such that a second script is started after one or more commands from the first script have been completed. For one embodiment, the system may include a default action script, such that upon the execution of any command or a defined set of commands, the default action script is triggered, unless an alternative action script is identified by the user.

[008] In order to make changes to the devices coupled to the network, the user simply changes data in the data store. Such a change in the data store triggers an automatic job generation, to generate an appropriate job to update the device(s) affected by the change in the data store. The control server then contacts the control point that controls the device(s) affected by the change, using a doorbell mechanism. This triggers the set-up of a secure communications method between the control server and the control point. The data sent by the control server to the control point is a job, which includes a JobScript that delivers any appropriate changes to the device configuration. The term delivery driver may be used for JobScript. The job may further include references to files that are needed. This user-modifiable job is executed within the control point.

[009] For one embodiment, the control point caches any support files that have been previously used, such that only a single copy of any support file needs to be transferred to the control point, even if multiple devices are being reconfigured.

[010] The control point then contacts the device, at the preset maintenance window of that device. The control point has its own scheduler, which is used to schedule the jobs sent to it by the control server.

[011] The device is then successfully updated, and data regarding the update, including a transcript of the update process, is sent back to the control server. The data store also stores the previous state of each device, providing historical states and versioning for every device. This permits a user to, for example, review previous states, or revert to a previous state. The previous states of each device are saved in the device profile for that device as well as in the device configuration file. In this way, a user may review the exact

configuration of a particular device at any past point–using the device configuration file—as well as configure an alternative device to the exact same configuration, using the device profile.

[012] Users have full control of configuration via device profiles, templates, and other means. One of the other means present is "service modules." Service modules permit functionality to be defined and "plugged in" into the control server. Users and service modules can insert information into the data store to control the various devices on the network. Users and service modules can define "new types" of data, i.e. arbitrary attributes in the data store. The service modules may define any types of actions or functions that may alternatively be individually executed by a user.

[013] Generating new control points is also simple. The control server generates a localized version of the control point, which is then installed on a machine. Localization creates a control point that knows which control server it belongs to. IP localization takes place when a passphrase is entered into the control point, as will be discussed below. For another embodiment, a dedicated appliance may be used. A passphrase is then generated. For one embodiment, the passphrase is a set of English words, of a specific length, such as four letters. This permits easy entry by user, and reduces the likelihood of mistyping. The passphrase includes, encoded within it, a one-time key or nonce, which then used by the newly set up control point to check in with the control server. The control server and control point then finish a key exchange process, and in future communication use a public key based communication.

[014] Figure 1 is a block diagram of one embodiment of a network with control servers, control points, and devices. The control server 110 provides

network control functions, and interacts with the control points 120 (A-D). The control server 110 provides a user interface, which allows the administrator to set up and manage control points 120, and manage the devices 130 (A-D) on the network through control points 120.

[015] The control server 110 may be coupled to the control point 120 through an insecure network 140. However, a secure communication channel 170 is established between the control server 110 and the control point 120. The secure communication channel may be secure sockets layer (SSL), secure shell (SSH), or any other secure communication protocols or mechanisms. One embodiment of establishing this secure communication method is described below. Alternatively, control server 110 may be coupled to control point 120 through a direct, secure connection.

[016] Control points 120 are coupled to at least one device 130. Control points 120 may be coupled to a device through a direct connection, through a secure network 160, through an insecure network, or through a Virtual Private Network 150. Note that multiple control points 120 may control the same device. The control server 110 ensures that only one control point 120 talks to any device 130 at the same time. The control point 120 need not know which devices it is coupled to until it receives a job for that device.

[017] As will be described below in more detail, the control points 120 permit a distributed management of the entire network 100. Furthermore, since the communication between the control point 120 and control server 110 is secure, overall security is enhanced.

[018] Figure 2 is one embodiment of computer system on which the present invention may be implemented. Figure 2 illustrates a typical data

processing system upon which one embodiment of the present invention is implemented. It will be apparent to those of ordinary skill in the art, however that other alternative systems of various system architectures may also be used.

[019] The data processing system illustrated in Figure 2 includes a bus or other internal communication means 245 for communicating information, and a processor 240 coupled to the bus 245 for processing information. The system further comprises a random access memory (RAM) or other volatile storage device 250 (referred to as memory), coupled to bus 245 for storing information and instructions to be executed by processor 240. Main memory 250 also may be used for storing temporary variables or other intermediate information during execution of instructions by processor 240. The system also comprises a read only memory (ROM) and/or static storage device 220 coupled to bus 240 for storing static information and instructions for processor 240, and a data storage device 225 such as a magnetic disk or optical disk and its corresponding disk drive. Data storage device 225 is coupled to bus 245 for storing information and instructions.

[020] The system may further be coupled to a display device 270, such as a cathode ray tube (CRT) or a liquid crystal display (LCD) coupled to bus 245 through bus 265 for displaying information to a computer user. An alphanumeric input device 275, including alphanumeric and other keys, may also be coupled to bus 245 through bus 265 for communicating information and command selections to processor 240. An additional user input device is cursor control device 280, such as a mouse, a trackball, stylus, or cursor direction keys coupled to bus 245 through bus 265 for communicating direction information

and command selections to processor 240, and for controlling cursor movement on display device 270.

[021] Another device that may optionally be coupled to computer system 230 is a communication device 290 for accessing other nodes of a distributed system via a network. The communication device 290 may include any of a number of commercially available networking peripheral devices such as those used for coupling to an Ethernet, token ring, Internet, or wide area network. Note that any or all of the components of this system illustrated in Figure 2 and associated hardware may be used in various embodiments of the present invention.

[022] It will be appreciated by those of ordinary skill in the art that any configuration of the system may be used for various purposes according to the particular implementation. The control logic or software implementing the present invention can be stored in main memory 250, mass storage device 225, or other storage medium locally or remotely accessible to processor 240. Other storage media may include floppy disks, memory cards, flash memory, or CD-ROM drives.

[023] It will be apparent to those of ordinary skill in the art that the methods and processes described herein can be implemented as software stored in main memory 250 or read only memory 220 and executed by processor 240. This control logic or software may also be resident on an article of manufacture comprising a computer readable medium having computer readable program code embodied therein and being readable by the mass storage device 225 and for causing the processor 240 to operate in accordance with the methods and teachings herein.

[024] The software of the present invention may also be embodied in a dedicated appliance containing a subset of the computer hardware components described above. For example, the dedicated appliance may be configured to contain only the bus 245, the processor 240, and memory 250 and/or 225. The handheld device may also be configured to include a set of buttons or input signaling components with which a user may select from a set of available options. The dedicated appliance may also be configured to include an output apparatus such as a liquid crystal display (LCD) or display element matrix for displaying information to a user of the dedicated appliance. Conventional methods may be used to implement such a dedicated appliance. The implementation of the present invention for such a device would be apparent to one of ordinary skill in the art given the disclosure of the present invention as provided herein.

[025] For one embodiment, control points 120 may be installed as an application on a standard personal computer. For one embodiment, the personal computer may become a dedicated system, which is only used as a control point 120. For one embodiment, the control point 120 runs over a hardened NetBSD operating system. The system may retain the ability to dual boot, into another operating system, such as Windows. For another embodiment, the device may be a dual use system, in which the control point 120 runs in the background, while the computer remains useable.

[026] Figure 3 is a block diagram of one embodiment of a control server. The control server 110 includes a scheduler 310, to schedule various interactions with control points (not shown), and to schedule maintenance windows for various devices. The control server 110 further includes a control point localizer

325, to generate new control points. For one embodiment, the control point localizer 325 may be used to generate the software for a localized control point on compact disk, floppy disk, or other media. This process is described in more detail with respect to Figures 7A and B.

[027] Control server 110 further includes control point updater 320, which updates control points. The control server 110 treats the control point as a normal device. Control point server 315 interacts with the control point API 317, to send data to the control points and receive data from the control points. Generally, the data sent to the control points is a job or a requested file or other job relevant data, which defines certain activities to be performed by the control point within a certain maintenance window. The data returned by the control point is a report of status, i.e. whether the execution of the job was successful, and any accompanying information.

[028] The control server 110 further includes schema translator 350 which interacts with the database API 355, which permits access to data store 360. For one embodiment, the data store is a SQL database. Alternatively, other types of data stores may be used. The database API 355 translates requests for the data store into the data store's interface language. The schema translator 350 presents the contents of the data store 360 in a hierarchical format. This permits logical interaction with the data store using command line interface 395, graphical user interface (GUI) 390, service modules, or business system 385.

[029] The command line interface 395 allows the user to execute complex commands composed together via pipes. Thus, a complex set of commands, where the output of one command is designed to be the input to the next command, may define any action within data store 360. Similarly, the GUI 390,

web based front-end, and/or business system 385 may permit the construction of complex commands that allow the user to make complex changes to the data store.

[030] Scheduler 310 schedules the maintenance windows for each device, and access timing for each control point. The process of scheduling is described in more detail below.

[031] Control server 110 further includes a device module API, to interact with device modules 345. Device modules are used to generate jobs for devices. The device module is described in more detail below, in Figure 11.

[032] The client API permits access to the control server by various service modules. The service modules 370 define complex workflow or known relationship types between devices. For example, a service module 370 may be used for initial element activation. Initial element activation may include configuring and activating a DSLAM, a CPE, and various other elements. The order in which these elements are configured may be critical. The service module 370 tracks this data, and sends the data to be used to create the configuration information to the device modules 345 in the correct order, to force the correct configuration.

[033] The service module 370 may include a separate graphical user interface 371, or command line interface 372. For another embodiment, the CLI and/or GUI of the service module 370 may be integrated with the CLI and/or GUI of the control server 310. The service module 370 may further, or alternatively, include a service module data store component 399. The service module data store component 399 interacts with schema translator 350, to recognize a specific change (or set of changes) to activate a service module. For

example, a service module that defines a virtual private network (VPN) may have a data store component 399 to recognize if one of the devices within the VPN has changed its configuration. In that case, the other devices within the VPN must be alerted to this fact, to keep the VPN functioning. The service module 370 is activated by the data store component 399, and performs configuration analysis and job generation.

[034] Various business systems 385 including web based front ends may address the control server 110. The client API may include an authorization layer, to verify that a service module business system, or user interface interacting with the control server 110 is authorized to do so.

[035] Figure 4 is a block diagram of one embodiment of a control point. Control point 130 includes an installation 410. The installation is the control server-branded disk or other media, which is used to generate the control point 130. The control point 130 may be generated on a stand-alone device, such as a computer, a special purpose appliance, on the same device as the control server or some other server, or a client device that may be used for other purposes. The control point 130, for one embodiment, installs its own operating system. For one embodiment, for devices that are not dedicated devices, the device may be made dual-boot, such that the original operating system is maintained on one portion of the system, while the operating system on which the control point 130 is running is installed on a separate portion. For one embodiment, the control point 130 is run on a specially hardened NetBSD operating system 470. For one embodiment, the hardening of operating system 470 includes making it more stable, and increasing security.

[036] For one embodiment, the control point's installation 410 may be over a Windows operating system, or another type of operating system. For another embodiment, the control point's installation 410 may be over a Unix variant or Unix-like operating system, such as Linux, Solaris, or AIX.

[037] The self-update logic 420 in control point 130 manages the updating of the control point itself.

[038] The cache 430 stores firmware versions, configuration files, and support files, which are sent to the control point 130 as a job. This permits the sending of smaller jobs, since only a reference to the appropriate files needs to be made. Thus, if 100 devices are upgraded, the actual firmware upgrade file, which may be quite large, need only be sent once from the control server to the control point.

[039] Secure communication mechanism 440 uses a key exchange to communicate with control server 110 (not shown). The secure communication mechanism 440 further generates the private key for the control point 130. For one embodiment the secure communication mechanism 440 further uses an initial one-time password (nonce) to connect to the control server for the first time.

[040] Execution environment for delivery driver 450 is where the indicated processes/jobs are executed. The execution environment 450 also executes the JobScript within the job to interact with device 490 to update/configure/obtain information from the device 490, as indicated by the job.

[041] Scheduling logic 460 maintains the maintenance windows for each device that the control point 130 has a job for. Scheduling logic 460 further

maintains the control point in clock sync with the control server. This is useful because as long as the control server is kept in good time sync with the world, that the control points are also, which ensures predictable and correct Job execution scheduling. It is also useful because it permits simple verification of what action occurred at what time. This may be useful, for example, in an evidentiary situation. Scheduling logic 460 further maintains a timer, to have the control point 130 contact the control server on a regular basis, if no request for connection is received from the control server. This is described in more detail below with respect to Figures 7A and B.

[042] Figure 5 illustrates one embodiment of the hierarchical structure of the data store. The hierarchical layout of the data store creates a data store tree that is simply configured. Statements 510 include four statements defining various data within the data store. For example, statement 1 identifies a device (myrouter), having an interface (fa0), having a particular address. As can be seen in tree 520, the device identification branch includes myrouter, and otherrouter, while the interface identification includes interface fa0, which further includes the particular address identified in the above statement.

[043] Figure 6A is a flowchart of one embodiment of installing a control point, from the control server's perspective. Figures 6A and 6B show one embodiment of the process of installing and starting up a new control point from the perspective of the control server and the control point. Figure 6A illustrates the control server's perspective.

[044] The process starts at block 605. A key pair is generated for the control server 610, and a generic control point executable is branded with the control server's public key, at block 615. These two steps take place once, when

the control server is initialized. For one embodiment, by branding the control point executable with the public key, the control point is permanently associated with a particular control server. For one embodiment, in order to move a control point from a first control server to a second control server, it must be reinstalled, with the branded executable of the second control server. Alternatively, the control point may be told in some trusted way about the other control server. For another embodiment, the second control server must have the same public/private key pair as the original control server. This increases security.

[045] At block 620, IP localization data is received for a new control point. IP localization data defines the future IP address, network mask, and gateway address of the new control point. This data is received from a user, or from an external program.

[046] At block 625, a passphrase is generated. The passphrase includes a unique one-time password. The passphrase further may include the IP localization data for the control point. For one embodiment, the passphrase comprises a set of words in the English language. This simplifies memorization and communication of the passphrase, and makes typos less likely. For one embodiment, the words are each four-letter words. Of course, the dictionary used for this may be changed, to any other language. Thus, the passphrase may be in any language, may have any number of words, and may have words of any particular length, or of a variety of lengths. This permits the use of a more complex passphrase that is still useable by a normal user.

[047] The passphrase is then displayed to the user, and someone installs the control point, and enters the phrase to activate the control point. Because the passphrase is words in a language the users speak, it can be easily communicated

over the telephone or another link to a remote administrator or user who actually performs the activation.

[048] At block 635, the connection from the new control point--recently installed by the user--is received at the control server. The connection uses the one-time password provided with the passphrase.

[049] At block 640, the control server determines whether the control point described by the passphrase is valid and inactive. The passphrase includes a control point number encoded within it. During the initial connection, the control point uses this number. If the number is invalid—i.e. is not in the control server database--, or the control point corresponding to the number is already active, this onetime password is not useable. In that case, the process continues to block 648 and ends without initializing the new control point.

[050] If the passphrase is valid, and the control point described by the passphrase has not been seen before, the process continues to block 642.

[051] At block 642, the control point's public key is received, and the control point is activated. The control point is added to the list of control points that can be used to control devices. At block 644, the frequency of contact with this control point is established. For one embodiment, the control point is contacted by the control server on a regular basis. However, if the control server has not been in touch with a control point within a preset period of time (set at this point), the control point initiates contact. At block 648, the process ends.

[052] At this point, the control server is aware of the control point, and can use the control point to contact various devices that may be reached through control point.

[053] Figure 6B is a flowchart of one embodiment of installing a control point, from the control point's perspective. Figure 6B illustrates the control point's perspective. The process starts at block 650.

[054] At block 655, the localized control point executable is installed on a system. Localization in this context refers to localization for the particular control server that will be the control server for this control point. The term "branding" also refers to the fact that the control point will only be useable by a particular control server that it is localized by. For one embodiment, the control point is installed by a human. As discussed above, the device may be a standard personal computer, which may be used as a dedicated control point or may be a dual-function system. For another embodiment, the control point may be a special function appliance, having only those elements needed to function as a control point.

[055] At block 657, the process receives the passphrase from the installing human. The passphrase encodes within it the localization information for the control point, as well as a one-time password.

[056] At block 665, the control point is personalized. Personalization includes setting up the Internet Protocol (IP) settings. For one embodiment, this further includes having the control point test its network connection using the IP localization data from the passphrase.

[057] At block 670, a new key pair is generated for secure communication with the control server. This key pair is used by the control point from this point forward to securely communicate with the control server.

[058] At block 675, the process tests whether there are any jobs for any devices coupled to the control point. The jobs may be encoded within the

passphrase, or may be transferred alongside it. If there are jobs available, at block 680 devices for which there are jobs are initialized. For one embodiment, this is done within one of the maintenance windows of the device, as will be discussed in more detail below. The process then returns to block 685. If no jobs are available, the process continues directly to block 685.

[059] At block 685, secure communication is established with the control server, using the one-time key, and a key exchange. For one embodiment, the key exchange is a Diffie-Hellman key exchange using session keys. The underlying connection is established using the one-time password, or nonce. Using that underlying secure connection, a session key is exchanged, and used for further communication.

[060] At block 690, the public key of the control point is passed to the control server over the secure connection created in block 685. This public key, generated at block 670 above, is used for creating secure communication channels in the future. This type of public key based secure communication is known in the art.

[061] At block 695, various data is received from the control server. This data may include a maintenance schedule for the control point itself, and various jobs for various devices. The control point then disconnects from the secure communication, and the process ends at block 699.

[062] Figure 7A is a flowchart of one embodiment of communication between the control server and the control point. The process starts at block 705. This process is a continuous process, in which the control server determines whether it has anything to say to a control point, and then communicates with

the control point. Note that although certain processes are illustrated as loops, they may be interrupt driven. This is true for all of the processes shown.

[063] At block 710, the control server determines whether it has a job for a control point. If not, this process continues to wait and check, until a job is found.

[064] At block 715, the process determines whether it is sufficiently near the maintenance window of the job to send the job. In general, the job is sent shortly before it is to be executed. If not, the process returns to block 710, to evaluate all outstanding jobs. Alternatively, the job may be sent at a job send window, as specified by the request that generated the job. For example, jobs may be sent months in advance. Since the control point includes its own scheduler, it can take care of timing the update/configuration process as specified by the job.

[065] Note that the process shown by blocks 710 and 715 may be combined into a single query, of whether there is a job due to be sent. Furthermore, this process may be spawned only if there is at least one outstanding job. Alternatively, this may be an interrupt driven process, in which the processing of the control server is interrupted when the conditions are correct, i.e. there is a job for a control point and it is appropriately near the maintenance window allocated for the job.

[066] If it is sufficiently near the maintenance window for the job to send it, the process continues to block 720.

[067] At block 720, a doorbell message is sent to the control point, requesting contact. For one embodiment, the doorbell message is an empty UDP packet sent to a particular port of the control point. For another embodiment,

the doorbell message is an attempt by the control server to open a TCP connection to the control point, at a predefined port address. This indicates to the control point that a request for contact has been received.

[068] The process then returns to block 710. The process also continues to block. The process shown in blocks 710-720 is independent from the processes described below, in blocks 725-745.

[069] At block 725, the process determines whether a secure connection has been received from the control point. This monitoring is continuous, whether or not a request for communication has been sent. If no communication has been received, the process continues waiting for communication. This too may be an interrupt driven process.

[070] If communication has been received, the process continues to block 730. At block 730, the process determines whether there has been a communication with this particular control point within the previous preset period. For one embodiment, the preset period may be a minute. If there has been such a communication, the process continues to block 733, and the connection is dropped. The process then loops back to block 725, to await new communication. This prevents denial of service (DoS) attacks against the control server, since only one connection per minute may be originated from any control point. Thus, if a control point is compromised, it is still unable to take down the control server.

[071] If there has been no communication within the preset period, the process continues to block 735.

[072] At block 735, any jobs for the control point are sent to the control point as jobs. Such a job may include a JobScript, configuration elements and

configuration files, and supporting files. The JobScript, configuration, and supporting files may simply be identifiers, permitting the control point to fill in data from its cache, or request the needed data during the execution of the JobScript. This will be described in more detail below.

[073] At block 740, job statuses for previous jobs executed by the control point are received. Job status may include a transcript of communication between the control point and device, a result code, an explanation for failure to successfully execute the JobScript, and any accompanying information.

[074] The communication is then completed, at block 747, and the process returns to waiting for a secure communication from a control point.

[075] Figure 7B is a flowchart of one embodiment of the process of communication between the control server and control point, from the control point's perspective. The process starts at block 750. At block 752, a long timer is started to monitor a period since the server has last contacted the control point. A short timer is started whenever a connection is received from the server, to monitor whether the connections are too frequent.

[076] As described above, portions of this process may be interrupt driven. This permits the control point to be in power save mode, if no jobs are due, and no connections are received. In that instance, the system may have a "wake up on timer" to check whether any requests have come in. Alternatively, the system may have a "wake on LAN" setting, which automatically wakes up the control point whenever a network connection request is received.

[077] If there is no request for connection, the process continues to block 760.

[078] At block 760, the process determines whether it is time to contact the control server. The control point periodically contacts the control server, if no connection has been received within a preset period of time. For one embodiment, this time may be 15 minutes, an hour, or any other time frame. For example, a control point may be behind a firewall or a network address translation (NAT) device, and thus the control server may not be able to contact the control point. This process makes sure that the control point and control server communicate on a regular basis. If it is not yet time to communicate with the control server, the process loops back to block 755. Note that the processes shown in blocks 755 and 760 are independent, and may be executed in any order.

[079] If the control point has received a request from the control server, the process continues to block 765. At block 765, the process determines whether there has been a connection request within a preset period of time. If so, the control point doesn't acknowledge the connection request, i.e. the control point passively ignores the connection request at block 767, and the process returns to block 755. This prevents a denial of service (DoS) attack on the control point, since only one connection request per the preset period of time is acknowledged. For one embodiment, the preset period of time may be one minute.

[080] If no connection has been received within the preset period of time, the process continues to block 770. If, at block 760, the process determined that it was time to contact the control server, the process continued directly to block 770.

[081] At block 770, the control point establishes a secure session with the control server. For one embodiment, the establishing of the secure session uses the public/private key pairs of the control point and control server. Figure 7C

illustrates in more detail the process of establishing a connection with the control server.

[082] At block 772, the control point requests and receives jobs from the control server, and puts the jobs into a job queue. For one embodiment, the job queue is ordered by maintenance window, i.e. the time at which the job should be executed.

[083] At block 774, the job statuses for previous jobs that have been completed are returned to the control server. At block 776, the connection with the control server is closed, and the timers are reset. The process then returns to block 755, to test whether a connection request has been received from the control server.

[084] Figure 7C illustrates one embodiment of establishing connection to the control server, including control server failover, from the perspective of the control point. The process corresponds to block 770 of Figure 7B, establishing a secure session with a control server. The process starts at block 778.

[085] At block 780, the server pointer is set to zero, to point to server zero on the server list. The control point has a server list, which lists, in order, the control servers which are available to the control point. The server list has at least one server on it. However, for one embodiment, at least two servers should be on the control server list.

[086] At block 782, the failure counter is set to zero.

[087] At block 784, the process attempts to establish a secure session with the control server pointed to by the server pointer. Thus, if this is the first pass through, the pointer points to server zero (see block 780).

[088] At block 786, the process determines whether the connection to the server has been successfully established. If so, the process continues to block 788. At block 788, the standard connection procedures, described above with respect to blocks 772-776, are executed. If appropriate, a new server list is received. For one embodiment, a new server list may be provided if the control server reached is not server zero, or if the server(s) formerly on the list are no longer available, or no longer the preferred server. The process then ends at block 790.

[089] If the connection is not successfully established at block 786, the process continues to block 792. At block 792, the failure counter is incremented. The failure counter counts the number of unsuccessful connection attempts that have been made to this server.

[090] At block 794, the process determines whether the failure counter is above a maximum count. In general, the system performs a preset number of connection attempts. If the failure counter has not yet reached the maximum, the process returns to block 784, and the system once again attempts to contact the server. For one embodiment, there may be a built-in wait prior to reattempting the connection.

[091] If the failure counter is above the maximum, the process continues to block 796. At block 796, the server pointer is incremented, to point to the next server in the server list. For one embodiment, the server pointer loops to zero. Thus, for example for a control point that has three servers on the server list, the control point would initially attempt to connect to server zero, then server one, then server two. If server two is not available, the control point attempts to connect to server zero again. This process loops until the control point is able to

connect to one of the servers on the server list. For one embodiment, there may be a short wait, prior to reattempting the servers on the list (e.g. at loop-over).

[092] In this way, the process can gracefully failover to a new server, without requiring that the control point have any additional intelligence, beyond a server list provided by the control server, a server pointer, and a failure counter.

[093] Figure 8 is a flowchart of one embodiment of updating a device from a control point. The process starts at block 805. At block 810, the process determines whether the control point has a job for a device that is in its maintenance window. Each job has associated with it one or more maintenance windows, during which times the job may be performed. Thus, this process determines whether there are any jobs that are scheduled for the current time. This process uses the internal scheduling logic of the control point, which permits the control point to determine whether it is time to perform a job.

[094] If there are no jobs that are to be performed, the process returns to block 810, to wait. For one embodiment, this is an interrupt driven process in which, when a maintenance window for a device opens, the control point is interrupted. For another embodiment, the control point periodically wakes up, and tests whether there are any current jobs in the maintenance window. If there is a job in the maintenance window, the process continues to block 820.

[095] At block 820, the control point attempts to connect to the device. For one embodiment, the connection may be through a direct connection, such as a serial connection. For another embodiment, the control point may use the device credentials to connect to the device. The credentials may be a public key, a password, or another type of credential to access the device.

[096] For one embodiment, the credential is a password. The control point initially uses the last password for the device. If the last password fails, and a new password exists that is not identical to the last password, the control point may attempt to use the new password. If the new password fails, or there is no new password, the control point may attempt to use the factory default password for the device. Note that this process of successively attempting passwords is specified by the JobScript, and may be skipped if the JobScript does not wish to try any other password than the current password.

[097] At block 825, the process determines whether the connection was successful. If the connection was not successful, the process continues to block 830.

[098] At block 830, the process determines whether the connection was rejected. If the connection was rejected -- e.g. the control point connected to the device and provided a credential, and the device rejected the connection -- then there is a problem in the interaction between the control point and device. Therefore further attempts at connection would not be useful. Thus, the process directly continues to block 850, where the transcript of the failure and the result code is stored, for later transmittal to the control server. The process then ends at block 855.

[099] If, on the other hand, the connection failed but not as a result of a rejection, the process returns to block 820, to attempt to connect to the device again. For one embodiment, the process waits for a short timer to expire prior to re-attempting the connection. For one embodiment, after a set number of attempts, the process may declare the connection unsuccessful. The process then continues to block 850, to report on the attempted connections.

[0100] If the connection request was successful at block 825, the process continued to block 835. At block 835, the process determines whether the device is in the expected state. The expected state includes configuration, firmware, etc. If the device is not in the expected state, the process continues directly to block 833, setting the result code to error, and then to block 850, to store a report on the status of the device. If the device is in the expected state, the process continues to block 840. Note that this process is implemented in the JobScript, and may be skipped. Thus, for one embodiment, the process may continue directly from block 825 to block 840.

[0101] At block 840, the device is updated, configured, and/or audited as specified by the JobScript, using the configuration files and supporting files as needed. For one embodiment, the device may also be tested, to verify that the updating/reconfiguration/audit was successful. Auditing a device may include obtaining a copy of all the files or a subset of the files and/or data on the device, obtaining a checksum of the firmware files, or in another way determining the state of the system. For one embodiment, audit jobs may be generated on a regular basis automatically.

[0102] At block 845, the control point disconnects from the device, having successfully executed the changes specified by the job.

[0103] At block 850, the control point stores the results of the execution. For one embodiment, the results include a transcript of the interaction between the control point and the device, a result code (i.e. success or failure), an explanation of the result if the result code indicates a failure, and a time. The time allows the control server to track exactly what changes were made when to which device. For one embodiment, all control points are synchronized to the control server, to permit determination of the exact time when any update/configuration change occurred. For one embodiment, this synchronization is sub-second synchronization. The process then ends at block 855.

[0104] Figure 9 illustrates one embodiment of command flow-through from the data store to a device. The command line interface 395 permits a user to enter various changes to data store 360, show as message 910. As described above, command line interface 395 permits the use of complex, pipelined syntax. The communication shown as message 915 alerts the control server that a change has been made in the data store. For one embodiment, this is not an actual communication, but rather an automatic event triggered by the change. For one embodiment, the system may be set up to automatically generate changes in the device profile periodically. For example, periodic automatic audits may be set up, by setting up a batch file or other self-executing file that periodically triggers a "new audit needed" note, or expires an old audit validity tag, or in a similar way requires a new audit to take place. Such an automatic change triggers the same message 915 as a change made via the CLI or other UI.

[0105] The control server 110 then generates a job for the change, and sends the job to the control point at the next scheduled communication, shown as message 920. The job includes a JobScript, a data file, lists of supporting files, and a time to execute the job, i.e. maintenance window. The job is also pushed back to the data store, to keep the data store up-to-date, shown as message 940.

[0106] The control point 120 executes the job at the appointed time, and updates the device 130, shown as message 925. For one embodiment, if the control point determines if the identified supporting files are in the control point's cache. If the supporting files are not in the cache, the control point determines which supporting documents, firmware copies, or other items are needed. The control point 120 then requests the needed files, shown as message 923. This data is returned by the control server, shown as block 924. For one embodiment, the control point may suspend execution of the JobScript to request such files.

[0107] The control point 120 returns the results of the updating/configuration process to the control server, shown as block 935. The results are also added to the data store, shown as message 940. In this way, changes made by the user using CLI 395 flow through the data store to generate an actual change in the device 130.

[0108] Figure 10 illustrates one embodiment of generating the data sent by the control server to the control point, for command flow-through. The process is initiated by a change 1005 in the data store 1010, to a device profile 1020. This change may be made using the command line interface (CLI), a graphical user interface (GUI), or another type of interface, including an interface in a service module.

[0109] The device profile 1020 thus changed identifies a descriptive triple, which in turn identifies a master, controller and possibly other elements 1040. As discussed below, the descriptive triple 1030 points to one master and one controller, and may further point to templates, JobScripts, and other elements.

[0110] The device ID and device profile are passed to the controller, to generate data for the master. For one embodiment, the device profile data passed to the controller includes the original device profile, the new device profile, and a delta, indicating the change(s) in the device profile.

[0111] If a template is indicated by the device module 1040, the template is executed 1060, to generate configuration file(s) 1055. For one embodiment, if there is a template specified by the device profile, it overrides the template identified by the descriptive triple.

[0112] The configuration files 1055 and preprocessed data generated by the controller 1050 are passed to the master 1070. The master then decides the course of action. The master may determine that no job needs to be generated, since the changes made in the profile do not need to be propagated to the device. For one embodiment, all changes to the device profile trigger this process. For example, the change of a comment may trigger this process. The master may thus decide that the device need not be updated/changed/audited as a result of the change to the device profile. If, however the master decides that the device should be changed/updated/audited, a job is created.

[0113] The job includes a maintenance window, a list of required files, and a correspondence to the "real names" of the list of required files. Thus, the job may be quite small. If the control point cache has the files required by the job, then the job is all that needs to be sent. However, if the control point does not have one or more of the required files, it requests the files from the control server, when the job is being executed.

[0114] Figure 11 is a block diagram of a device module and a descriptive triplet embodying the device module. A Device Module 1110 is a list of name-to-filename associations. A Device Module 1110 must include a Controller 1135 and a Master 1120. The Controller 1135 preprocesses data that is derived from the Data Store, and the Master 1120 uses the data created by the Controller 1135 to create the Job. The device module 1110 may further include one or more templates 1125 and a JobScript 1130, if the Master 1120 uses templates and/or a JobScript.

[0115] The template 1125 takes data from controller 1135 and creates a configuration file and/or enough data for the JobScript 1130 to make the requested change to the device state. The controller 1135 manipulates the data from the device profile 1150 and other sources to create data for use by the templates 1125, in accordance with the modification indicated by the user/service module.

[0116] A template 1125 may invoke one or more templates, which may in turn invoke other templates. Similarly, controllers 1135 may invoke other controllers, and so on.

[0117] A descriptive triple 1150 points to the device module 1110. The descriptive triple 1150 includes a device model name 1155, a platform 1160, and a firmware revision 1165 for which this is the appropriate device module 1110. The descriptive triple 1150 may further include a template. The template referenced in the device profile 1150 overrides the template referenced in the device module 1110.

[0118] A plurality of descriptive triples 1150 may point to the same device module 1110. This generally occurs when the devices and firmware configurations are very similar. The descriptive triple 1150 defines the language which may be used to access the devices which are described the by descriptive triple 1150.

[0119] Note that the device module 1110 shown is an example only. A device module 1110 need only include a master 1120 and a controller 1135 to process data. The other elements may be added, if they are used by the controller 1135 and/or the master 1120.

[0120] Firmware is managed separately from Device Modules. The firmware file list 1170 is key/value pairs indexed on a double -- platform, and firmware revision.

[0121] Figure 12 illustrates one embodiment of the history/versioning process. A data store entry 1210 is shown. The data store entry 1210 can be any type of data, including configuration information, device profile information, transcripts, pointers to files (each file having a version), and pointers to template files. Each data store entry 1210 has a version number and a time/date stamp. The data store entry 1210 has associated with it historical data 1220, e.g. prior versions of the data.

[0122] Each change to the data store entry 1210 generates a new "current version" of the entry, and stores the previous version as historical data 1220. For one embodiment, the version number is incremented by one, and versions are integers. The time attached to each entry indicates when the change was made. In this way, the changes to the data store entry, to device profiles, templates, configurations, etc. can be tracked. For example, if a problem occurs at a certain time, all records changed just prior to that time may be examined to determine the source of the error. Furthermore, a previous configuration may be reconstructed, using this data store. Additionally, using the device profile data, appropriate configurations for different devices may be generated, using the data store.

[0123] The process described above permits control of a complex network. The flexibility and configurability of the system allows a user to alter the data structures used, alter the interfaces, and otherwise make the best use of this system.

[0124] In the foregoing specification, the invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention as set forth in the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.